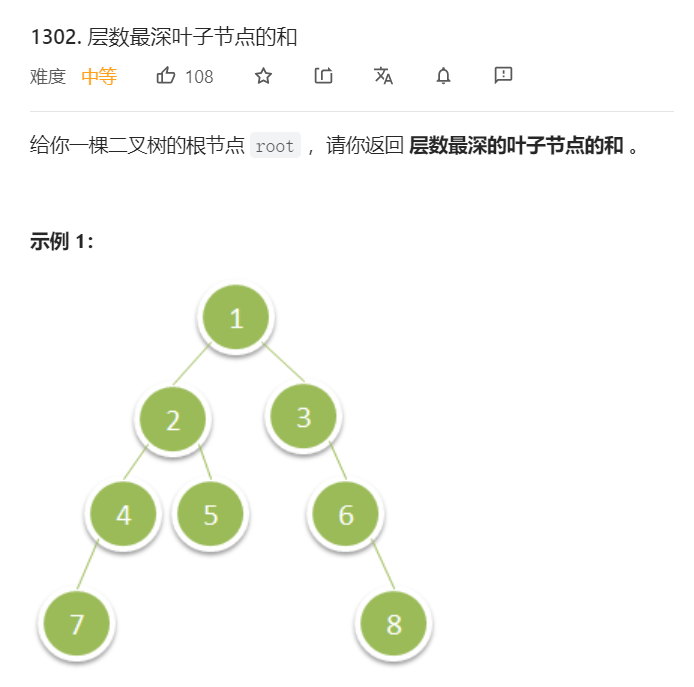
2022.08.17

今天的问题是一个有趣的二叉树递归问题，我看到了几个不同的解题思路，把他记录下来。



1. **层次遍历，BFS**

层次遍历是最直接思考到的想法，只要知道最底层的子叶，可以直接相加得出结果。

计算最深节点之和也可以使用广度优先搜索实现。使用广度优先搜索时，对二叉树层序遍历，此时不需要维护最大层数，只需要确保每一轮遍历的节点是同一层的全部节点，则最后一轮遍历的节点是全部最深节点。

初始时，将根节点加入队列，此时队列中只有一个节点，是同一层的全部节点。每一轮遍历时，首先得到队列中的节点个数size，从队列中取出 size 个节点，则这 size 个节点是同一层的全部节点，记为第 x 层。遍历时，第 x 层的每个节点的子节点都在第x+1 层，将子节点加入队列，则该轮遍历结束之后，第 x 层的节点全部从队列中取出，第x+1 层的节点全部加入队列，队列中的节点是同一层的全部节点。因此该方法可以确保每一轮遍历的节点是同一层的全部节点。

遍历过程中，分别计算每一层的节点之和，则遍历结束时的节点之和即为层数最深叶子节点的和。

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution {

public:

int deepestLeavesSum(TreeNode\* root) {

queue<TreeNode \*>q;

q.push(root);

int ans=0;

while(!q.empty())

{

int len=q.size();

ans=0;

while(len--)

{

TreeNode \* t=q.front();

q.pop();

ans+=t->val;

if(t->left)

q.push(t->left);

if(t->right)

q.push(t->right);

}

}

return ans;

}

};

有点难理解，但是我看懂了。每次pop或者push一个数字。

注意，Tree Node \* 是结构体指针，指向队列q。

1. **DFS遍历**

由于层数最深的节点一定是叶节点，因此只要找到所有层数最深的节点并计算节点值之和即可。

可以使用深度优先搜索实现。从根节点开始遍历整个二叉树，遍历每个节点时需要记录该节点的层数，规定根节点在第 0 层。遍历过程中维护最大层数与最深节点之和。

对于每个非空节点，执行如下操作。

1.判断当前节点的层数与最大层数的关系：

* 如果当前节点的层数大于最大层数，则之前遍历到的节点都不是层数最深的节点，因此用当前节点的层数更新最大层数，并将最深节点之和更新为当前节点值；
* 如果当前节点的层数等于最大层数，则将当前节点值加到最深节点之和。

2.对当前节点的左右子节点继续深度优先搜索。

遍历结束之后，即可得到层数最深叶子节点的和。

class Solution {

private:

int maxLevel = -1;

int sum = 0;

public:

int deepestLeavesSum(TreeNode\* root) {

dfs(root, 0);

return sum;

}

void dfs(TreeNode\* node, int level) {

if (node == nullptr) {

return;

}

if (level > maxLevel) {

maxLevel = level;

sum = node->val;

} else if (level == maxLevel) {

sum += node->val;

}

dfs(node->left, level + 1);

dfs(node->right, level + 1);

}

};